dabl

Automatic ML with a human in the loop

https://amueller.github.io/dabl

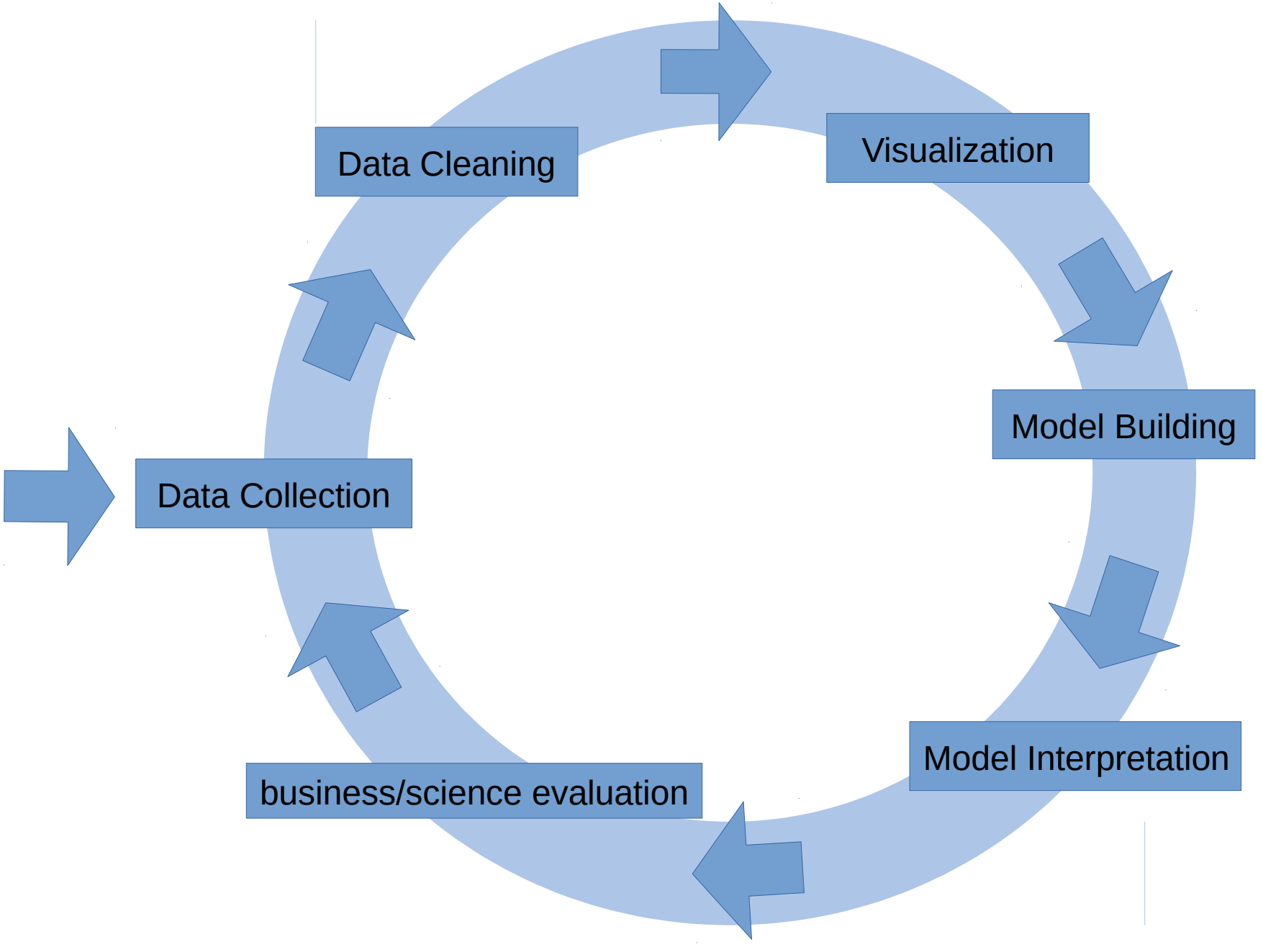
Andreas Müller

Associate Research Scientist
Columbia University
Scikit-learn Technical Committee

Alfred P. Sloan
FOUNDATION

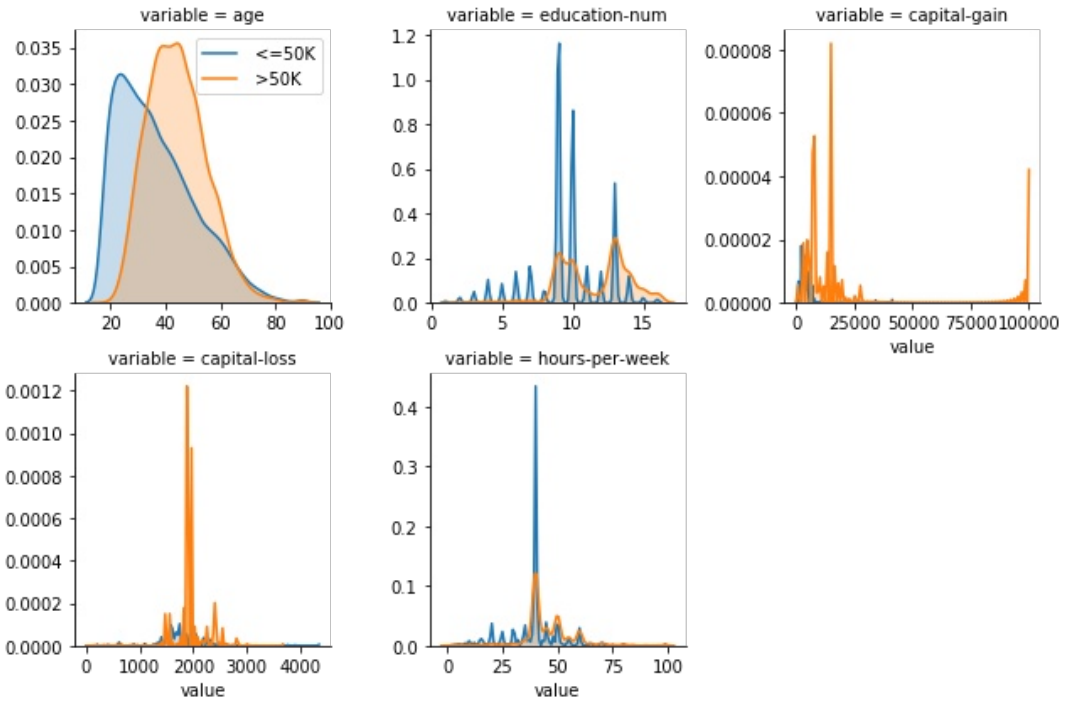
COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

A real world ML workflow



ML with sklearn & pandas

```
import pandas as pd
import seaborn as sns

data = pd.read_csv("adult.csv", index_col=0)

cols = data.columns[data.dtypes != object].tolist() + ['income']
df = data.loc[:, cols].melt("income")
g = sns.FacetGrid(df, col='variable', hue='income',
                  sharey=False, sharex=False, col_wrap=3)
g = g.map(sns.kdeplot, "value", shade=True)
g.axes[0].legend()
```



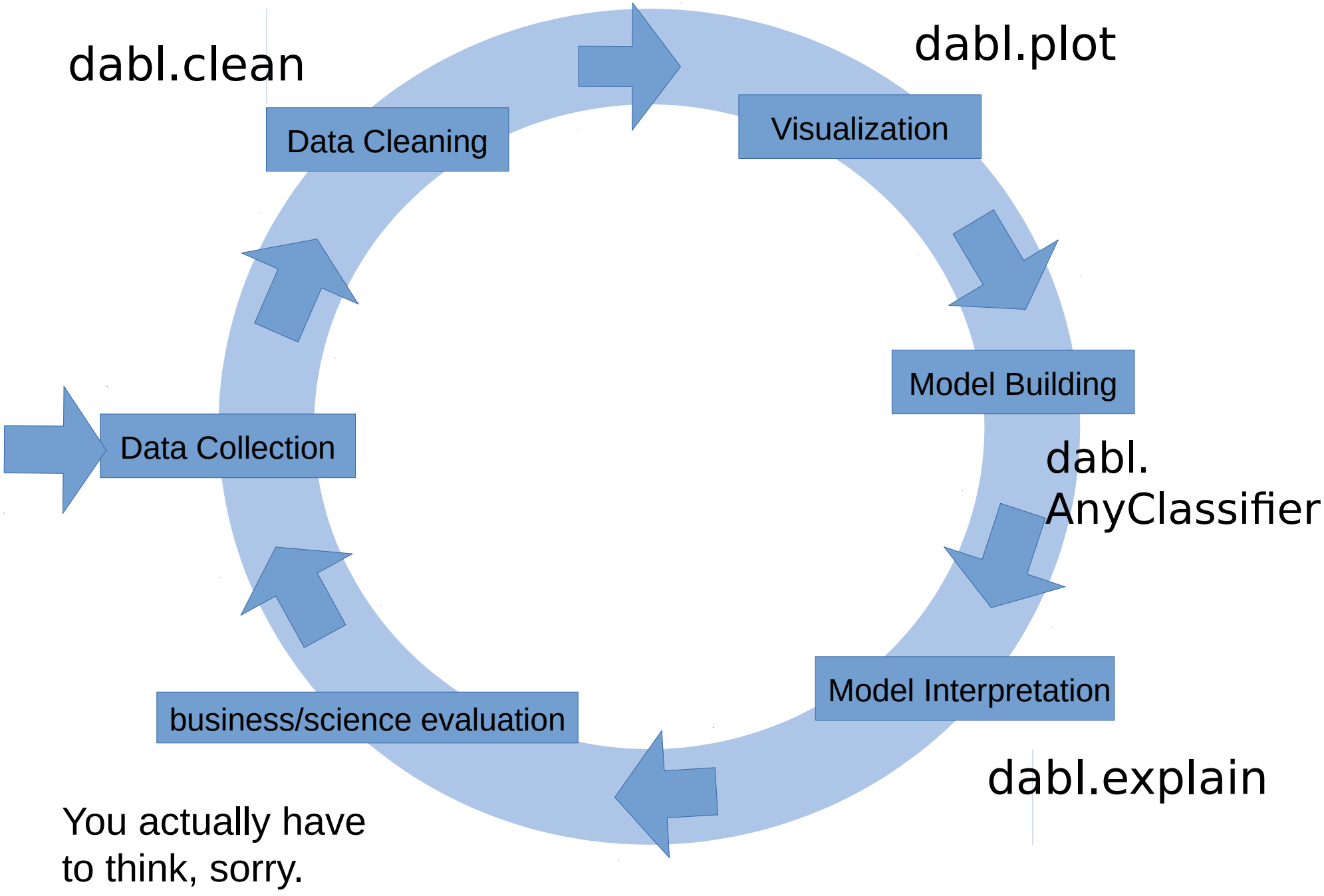
ML with sklearn & pandas

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

categorical_columns = data_features.dtypes == object

cont_pipe = Pipeline([('scaler', StandardScaler()),
                      ('imputer', SimpleImputer(strategy='median', add_indicator=True))])
cat_pipe = Pipeline([('ohe', OneHotEncoder(handle_unknown='ignore')),
                     ('imputer', SimpleImputer(strategy='most_frequent', add_indicator=True))])

pre = ColumnTransformer([('categorical', cat_pipe, categorical_columns),
                         ('continuous', cont_pipe, ~categorical_columns),
                        ])

model = Pipeline([('preprocessing', pre), ('clf', LogisticRegression())])
param_grid = {'clf__C': np.logspace(-3, 3, 7)}
grid_search = GridSearchCV(model, param_grid=param_grid)
grid_search.fit(X_train, y_train)
```


A NEW HOPE

data
analysis
baseline
library

A NEW HOPE


dabl.clean
dabl.plot
Data Cleaning
Visualization
Model Building
Data Collection
dabl.
AnyClassifier
Model Interpretation
business/science evaluation
dabl.explain
You actually have
to think, sorry.

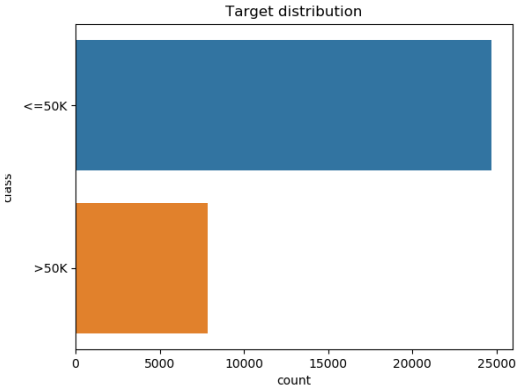
Data cleaning
& preprocessing

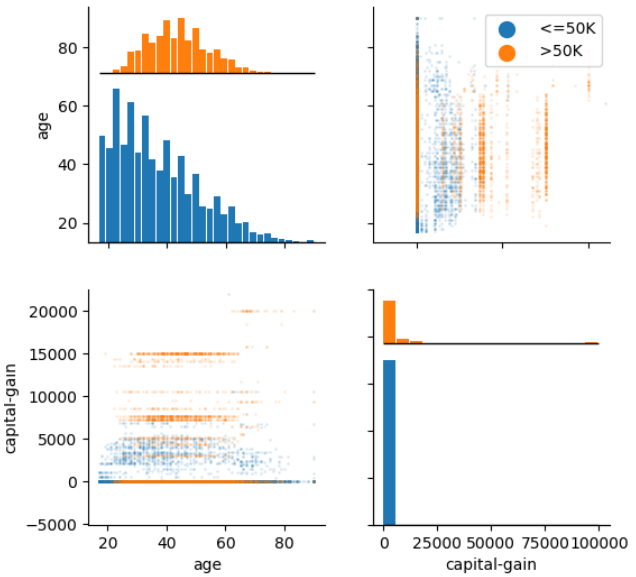
dabl.clean

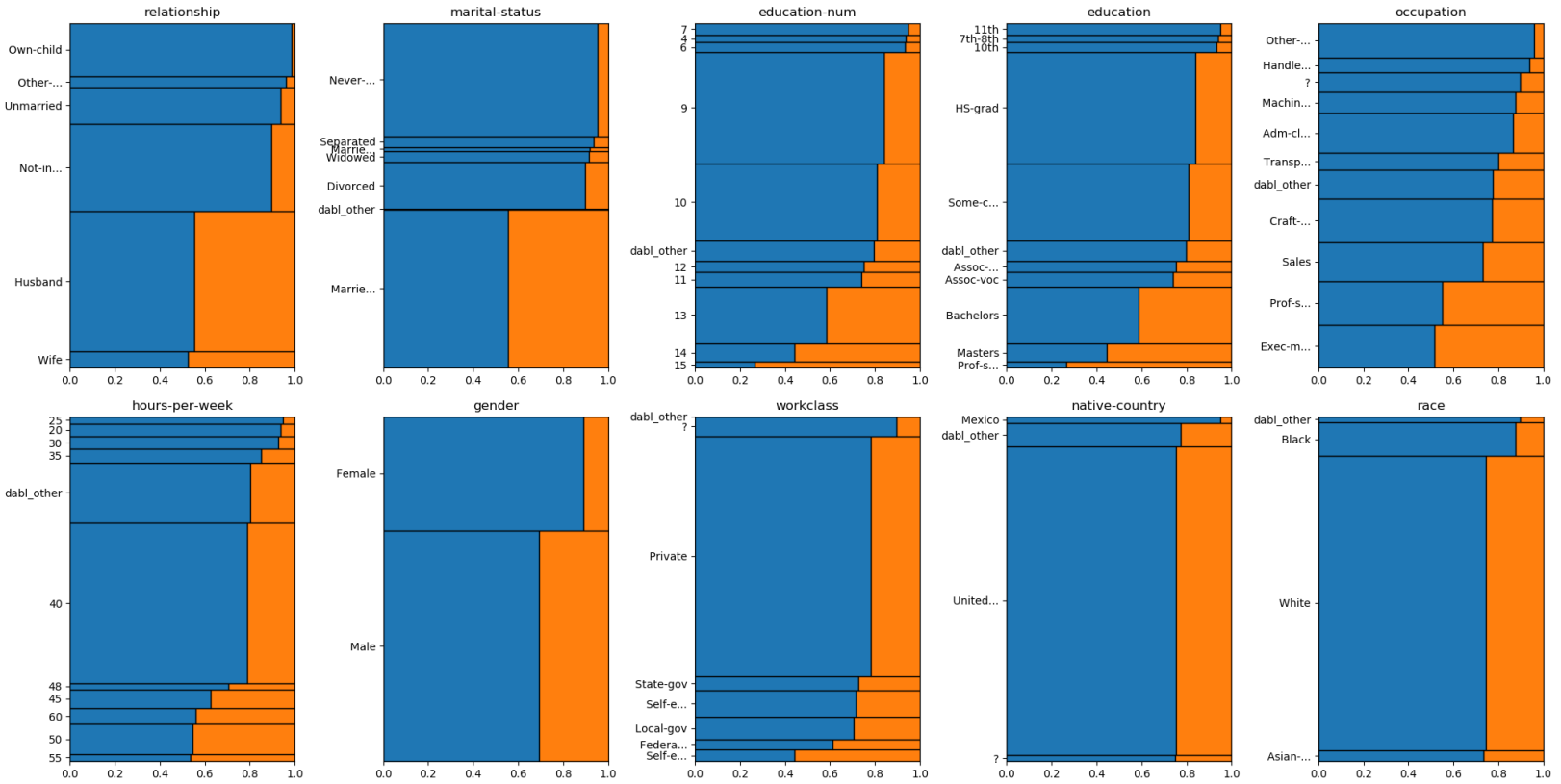
- Detect types (can overwrite)
- Detect Missing / rare values
- Detect ordinal vs categorical
- Detect near-constant
- Detect index

Visualization

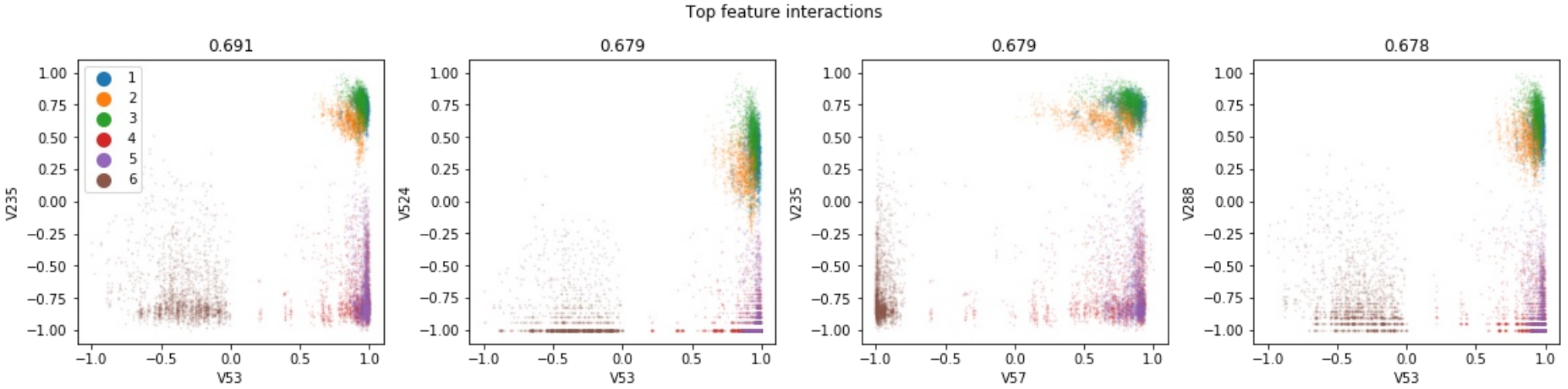
```
data = pd.read_csv("adult.csv", index_col=0)
plot(data, 'income', scatter_alpha=.1)
```



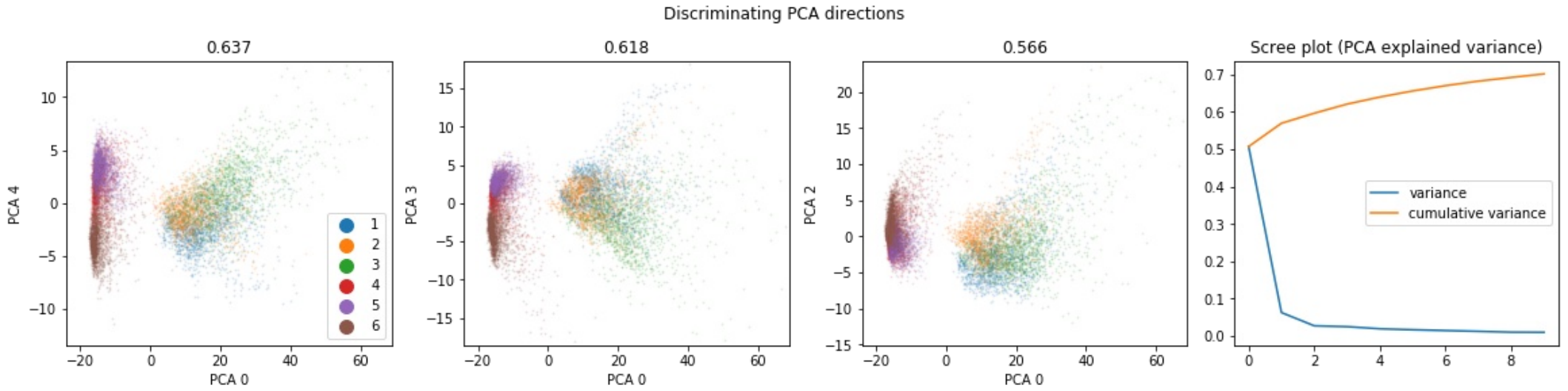




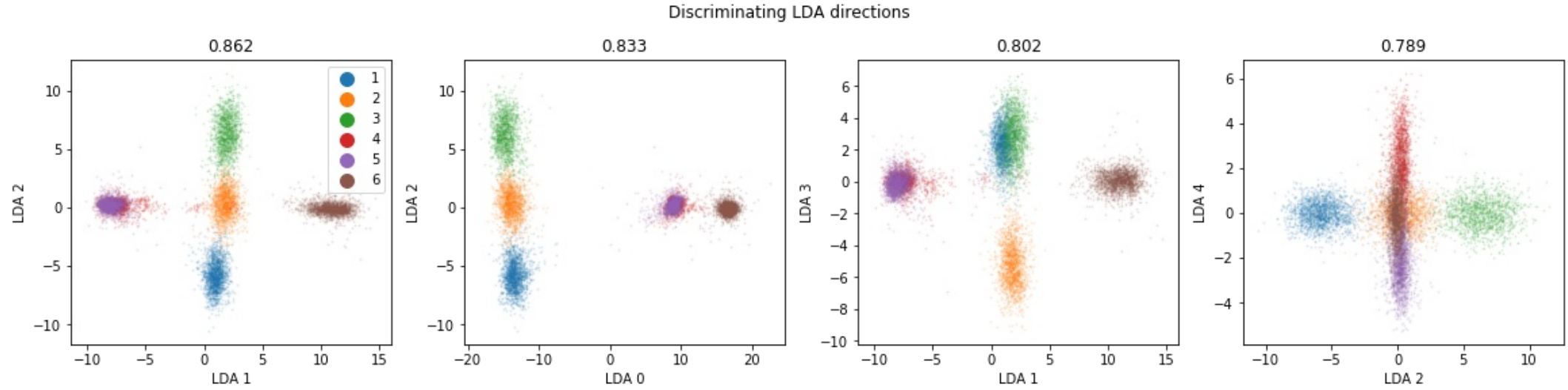
Pairwise Plots



Principal Component Analysis



Linear Discriminant Analysis



Preprocessing

```
X, y = ames_df.drop('SalePrice', axis=1), ames_df.SalePrice
ep =  EasyPreprocessor().fit(X, y)
```

```
/home/andy/checkout/dabl/dabl/preprocessing.py:258: UserWarning: Discarding near-constant
'Land Slope', 'Condition 2', 'Roof Matl', 'Heating', 'Low Qual Fin SF', 'Kitchen AbvGr',
rea', 'Misc Val']
  near_constant.index[near_constant].tolist()))
```

```
ep.ct_
```

```
ColumnTransformer(n_jobs=None, remainder='drop', sparse_threshold=0.1,
                  transformer_weights=None,
                  transformers=[('continuous',
                                 Pipeline(memory=None,
                                          steps=[('simpleimputer',
                                                  SimpleImputer(add_indicator=False,
                                                                copy=True,
                                                                fill_value=None,
                                                                missing_values=nan,
                                                                strategy='median',
                                                                verbose=0)),
                                                 ('standardscaler',
                                                  StandardScaler(copy=True,
                                                                 with_mean=True,
                                                                 with_std=True))],...
```

Simple Prototypes

```python
from dabl import SimpleClassifier
data = pd.read_csv("adult.csv", index_col=0)
SimpleClassifier().fit(data, target_col='income')
```

Either X, y
Or dataframe, target_col

```
/home/andy/checkout/dabl/dabl/preprocessing.py:258: UserWarning: Discarding near-constant featur
  near_constant.index[near_constant].tolist()))
Running DummyClassifier(strategy='prior')
accuracy: 0.759 average_precision: 0.241 f1_macro: 0.432 recall_macro: 0.500 roc_auc: 0.500
=== new best DummyClassifier(strategy='prior') (using recall_macro):
accuracy: 0.759 average_precision: 0.241 f1_macro: 0.432 recall_macro: 0.500 roc_auc: 0.500

Running GaussianNB()
accuracy: 0.407 average_precision: 0.288 f1_macro: 0.405 recall_macro: 0.605 roc_auc: 0.607
=== new best GaussianNB() (using recall_macro):
accuracy: 0.407 average_precision: 0.288 f1_macro: 0.405 recall_macro: 0.605 roc_auc: 0.607

Running MultinomialNB()
accuracy: 0.831 average_precision: 0.773 f1_macro: 0.787 recall_macro: 0.815 roc_auc: 0.908
=== new best MultinomialNB() (using recall_macro):
accuracy: 0.831 average_precision: 0.773 f1_macro: 0.787 recall_macro: 0.815 roc_auc: 0.908

Running DecisionTreeClassifier(class_weight='balanced', max_depth=1)
accuracy: 0.710 average_precision: 0.417 f1_macro: 0.682 recall_macro: 0.759 roc_auc: 0.759
Running DecisionTreeClassifier(class_weight='balanced', max_depth=5)
accuracy: 0.784 average_precision: 0.711 f1_macro: 0.750 recall_macro: 0.811 roc_auc: 0.894
Running DecisionTreeClassifier(class_weight='balanced', min_impurity_decrease=0.01)
accuracy: 0.718 average_precision: 0.561 f1_macro: 0.693 recall_macro: 0.779 roc_auc: 0.848
Running LogisticRegression(C=0.1, class_weight='balanced')
accuracy: 0.819 average_precision: 0.789 f1_macro: 0.783 recall_macro: 0.832 roc_auc: 0.915
=== new best LogisticRegression(C=0.1, class_weight='balanced') (using recall_macro):
accuracy: 0.819 average_precision: 0.789 f1_macro: 0.783 recall_macro: 0.832 roc_auc: 0.915
```

Automatic Model Search

Successive Halving on a fixed, diverse portfolio:

```
from sklearn.model_selection import train_test_split
df_train, df_test = train_test_split(data)
ac = AnyClassifier().fit(df_train, target_col='target')
```

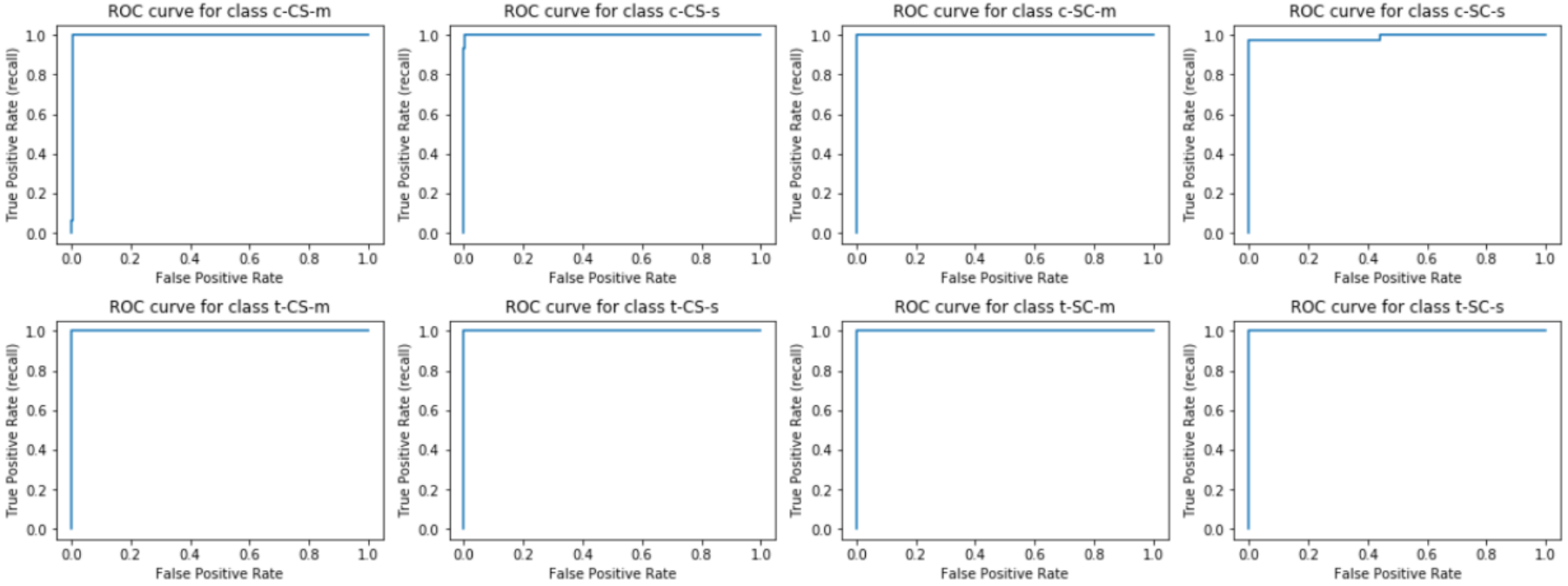
```
from sklearn.model_selection import train_test_split
df_train, df_test = train_test_split(data)
ac = AnyClassifier().fit(df_train, target_col='target')
```

```
import dabl
dabl.explain(ac, X_val=df_test, target_col='target')
```

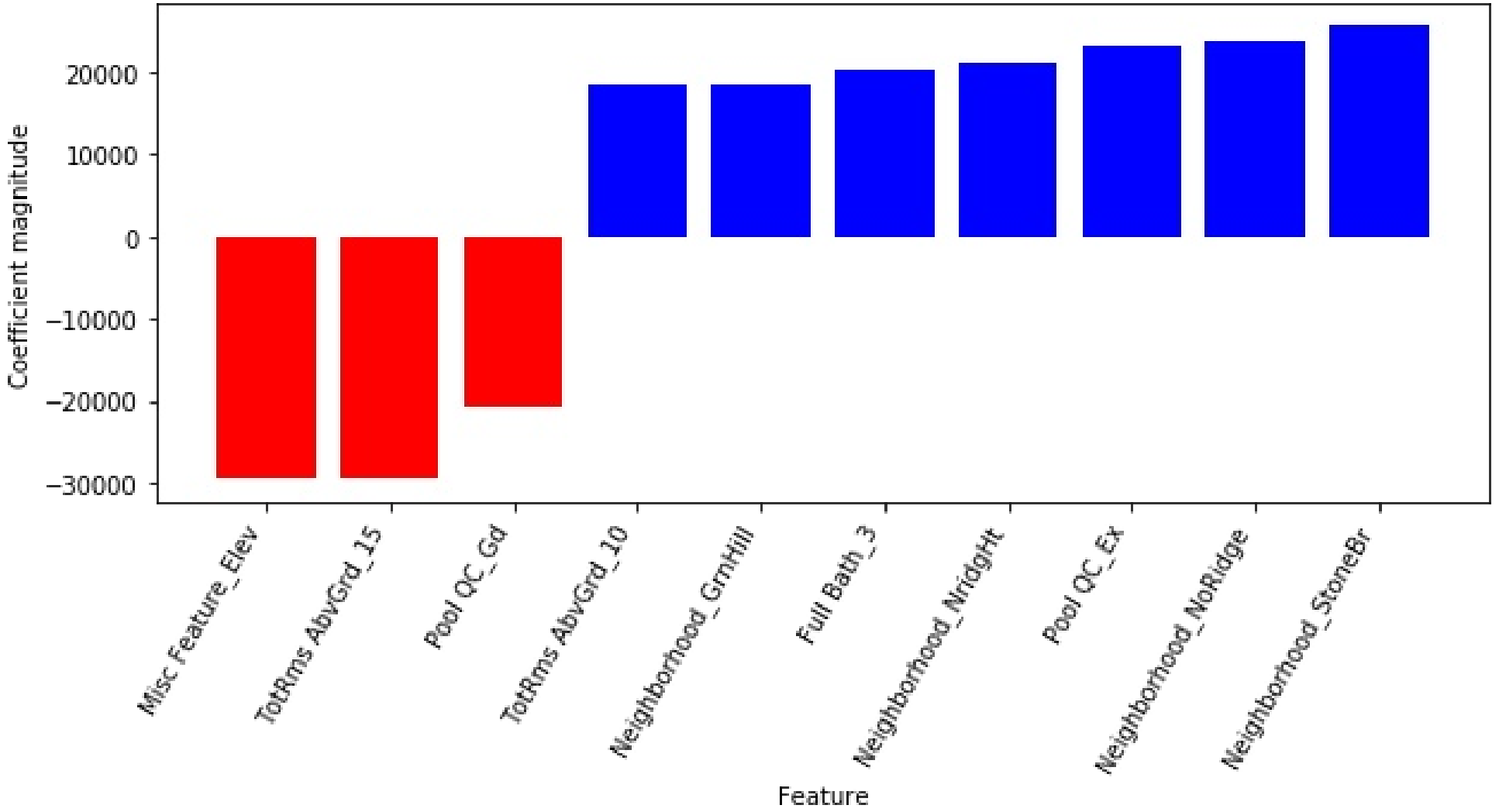
Metrics

	precision	recall	f1-score	support
c-CS-m	0.97	0.97	0.97	32
c-CS-s	0.94	1.00	0.97	31
c-SC-m	1.00	1.00	1.00	37
c-SC-s	1.00	0.94	0.97	34
t-CS-m	1.00	0.98	0.99	41
t-CS-s	1.00	1.00	1.00	21
t-SC-m	1.00	1.00	1.00	33
t-SC-s	0.98	1.00	0.99	41
accuracy			0.99	270
macro avg	0.99	0.99	0.99	270
weighted avg	0.99	0.99	0.99	270

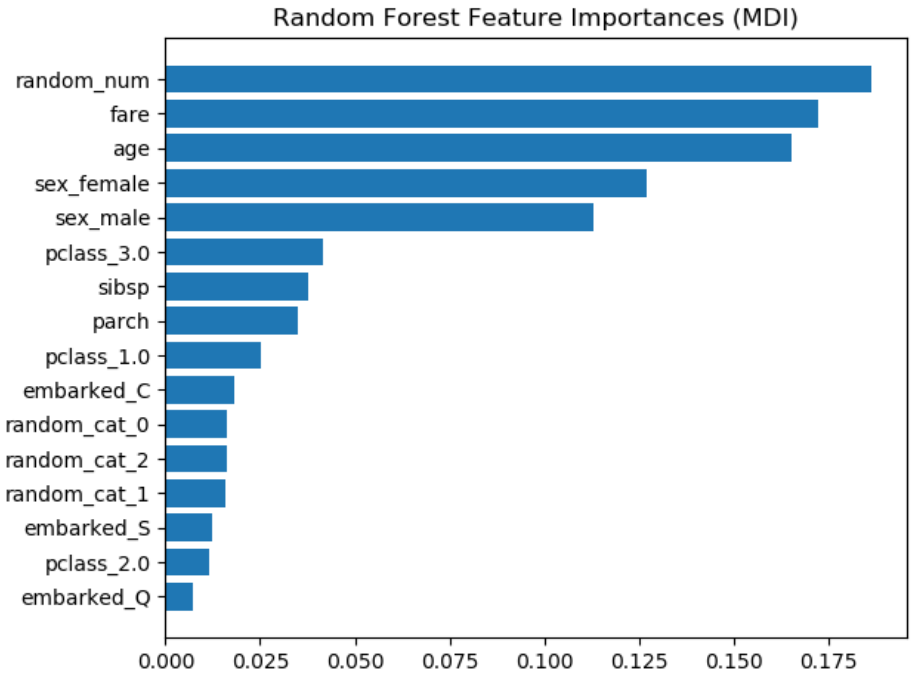
```
[[31  1  0  0  0  0  0  0]
 [ 0 31  0  0  0  0  0  0]
 [ 0  0 37  0  0  0  0  0]
 [ 1  0  0 32  0  0  0  1]
 [ 0  1  0  0 40  0  0  0]
 [ 0  0  0  0  0 21  0  0]
 [ 0  0  0  0  0  0 33  0]
 [ 0  0  0  0  0  0  0 41]]
```

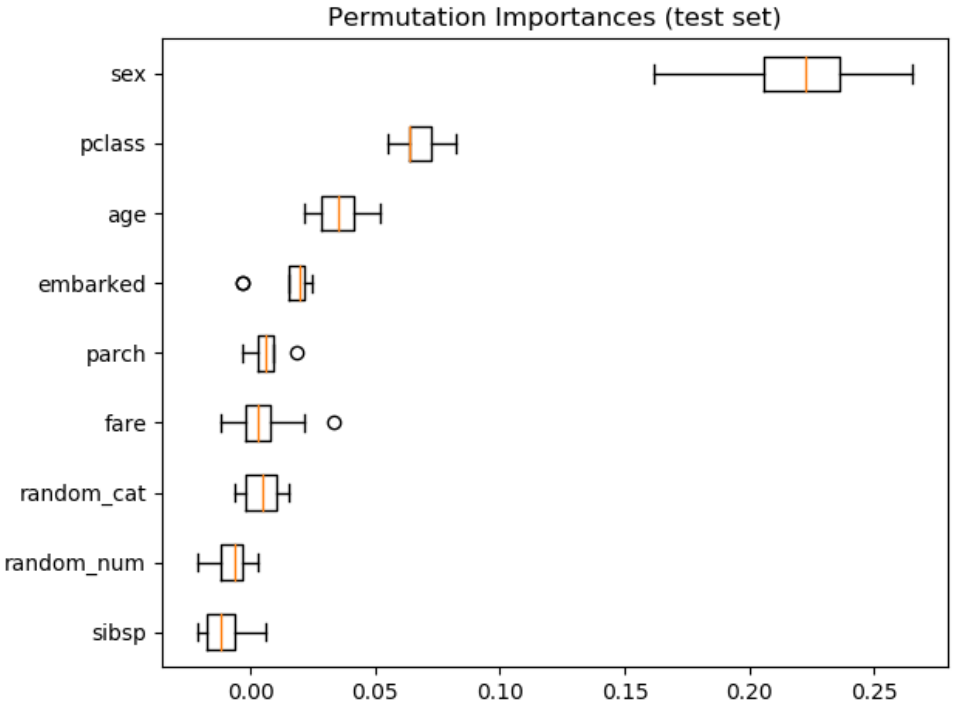


Coefficients / Feature importances



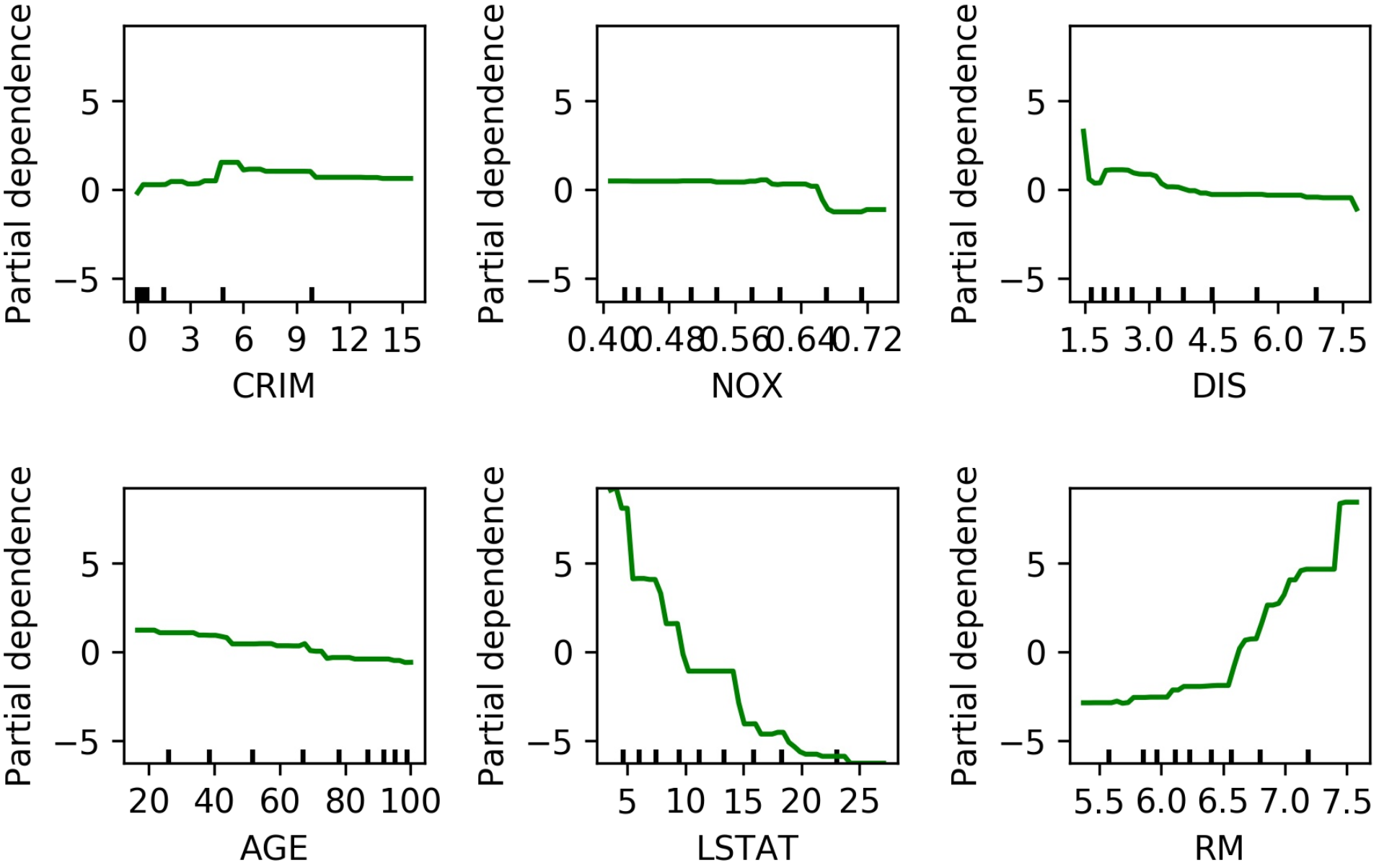
Permutation Importance





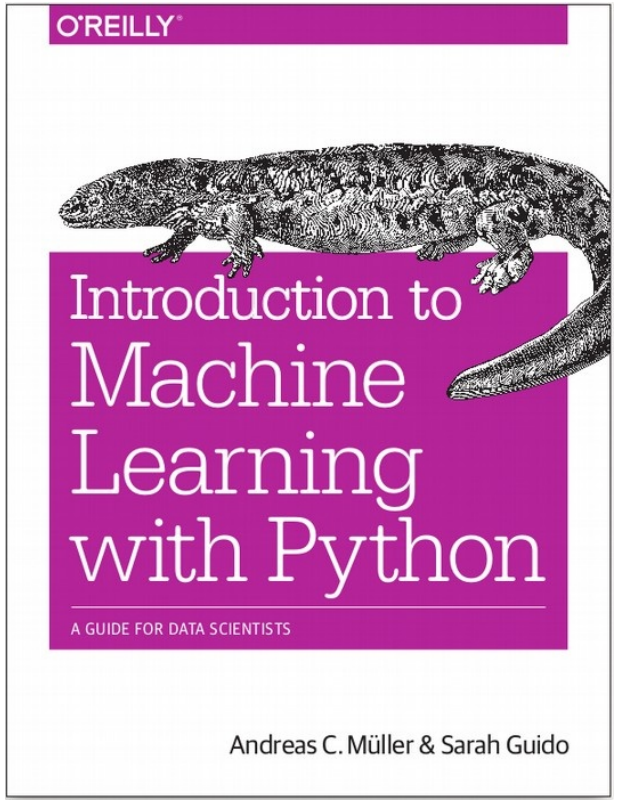
https://scikit-learn.org/dev/auto_examples/inspection/plot_permutation_importance.html

Partial Dependence Plots



$ pip install dabl

https://amueller.github.io/dabl



amueller.github.io

@amuellerml

@amueller

andreas.mueller
@columbia.com